Symantec™ Protection Engine for Network Attached Storage 8.1 C SDK Guide

Symantec

Symantec™ Protection Engine for Network Attached Storage 8.1 C SDK Guide

Legal Notice

Copyright © 2019 Symantec Corporation. All rights reserved.

Symantec, the Symantec Logo, the Checkmark Logo and are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.

This Symantec product may contain third party software for which Symantec is required to provide attribution to the third party (“Third Party Programs”). Some of the Third Party Programs are available under open source or free software licenses. The License Agreement accompanying the Software does not alter any rights or obligations you may have under those open source or free software licenses. Please see the Third Party Legal Notice Appendix to this Documentation or TPIP ReadMe File accompanying this Symantec product for more information on the Third Party Programs.

The product described in this document is distributed under licenses restricting its use, copying, distribution, and decompilation/reverse engineering. No part of this document may be reproduced in any form by any means without prior written authorization of Symantec Corporation and its licensors, if any.

THE DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. SYMANTEC CORPORATION SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.

The Licensed Software and Documentation are deemed to be commercial computer software as defined in FAR 12.212 and subject to restricted rights as defined in FAR Section 52.227-19 "Commercial Computer Software - Restricted Rights" and DFARS 227.7202, et seq. "Commercial Computer Software and Commercial Computer Software Documentation," as applicable, and any successor regulations, whether delivered by Symantec as on premises or hosted services. Any use, modification, reproduction release, performance, display or disclosure of the Licensed Software and Documentation by the U.S. Government shall be solely in accordance with the terms of this Agreement.

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

https://www.symantec.com

Symantec Support

All support services will be delivered in accordance with your support agreement and the then-current Enterprise Technical Support policy.

Knowledge Base Articles and Symantec Connect

Before you contact Technical Support, you can find free content in our online Knowledge Base, which includes troubleshooting articles, how-to articles, alerts, and product manuals. In the search box of the following URL, type the name of your product:

https://support.symantec.com

Access our blogs and online forums to engage with other customers, partners, and Symantec employees on a wide range of topics at the following URL:

https://www.symantec.com/connect

Technical Support and Enterprise Customer Support

Symantec Support maintains support centers globally 24 hours a day, 7 days a week. Technical Support's primary role is to respond to specific queries about product features and functionality. Enterprise Customer Support assists with non-technical questions, such as license activation, software version upgrades, product access, and renewals.

For Symantec Support terms, conditions, policies, and other support information, see:

https://entced.symantec.com/default/ent/supportref

To contact Symantec Support, see:

https://support.symantec.com/en_US/contact-support.html

Contents

Symantec Support 4

Chapter 1 Getting started 7

About Symantec Protection Engine for Network Attached Storage 7
About the software developer's guide 8
What's new in Symantec Protection Engine 8.1 8
About integrating with Symantec Protection Engine 8
About ICAP 9
About the C API 10
About other protocols 10
Enhanced categorization of threats 10
Enabling enhanced categorization of threats 12
About backward compatibility 14
Better handling of Unscannable files 14
About Keep-Alive connection support 14

Chapter 2 Constructing clients using the antivirus client-side API library 16

General procedure for scanning 16
Compiling and linking 17
Compiling on Windows Server 2008 R2 x64 17
Solaris 19
Red Hat Linux 19
Exceptions and error handling 20
API functions 20
File-based scanning 21
Stream-based scanning 21
ScanClientStartUp 21
ScanClientSecureStartUp 24
ScanClientScanFile 28
ScanResultGetNumProblems 34
ScanResultGetNumProblemsEx 35
ScanResultGetProblem 37
SC_DECODE_DISPOSITION 45

ScanResultsFree 46
ScanClientShutDown 46
ScanClientSecureShutDown 47
ScanClientStreamStart 47
ScanClientStreamSendBytes 51
ScanClientStreamFinish 52
ScanClientStreamAbort 54
ScanGetNumConnectErrors 55
ScanGetConnectError 56

Appendix A Using the antivirus API 58

About the sample code 58
Sample code (using Legacy connection format) 58

Index 74

Chapter 1

Getting started

This chapter includes the following topics:

- About Symantec Protection Engine for Network Attached Storage
- About the software developer's guide
- What's new in Symantec Protection Engine 8.1
- About integrating with Symantec Protection Engine
- Enhanced categorization of threats
- Better handling of Unscannable files
- About Keep-Alive connection support

About Symantec Protection Engine for Network Attached Storage

Symantec™ Protection Engine for Network Attached Storage is hereafter referred to as Symantec™ Protection Engine.

Symantec™ Protection Engine is a carrier-class, network-accessible, and content-scanning engine. Symantec Protection Engine provides content scanning capabilities to any application on an IP network, regardless of platform.

Symantec Protection Engine features all of the key content-scanning technologies that are available in the complete line of Symantec products. Symantec Protection Engine is one of the most effective solutions available for protecting your network against a variety of undesirable content.

For more information, see the *Symantec Protection Engine Implementation Guide*.

About the software developer's guide

Software developers can use the information that is provided in the *Symantec Protection Engine Software Developer's Guide* to create client applications that let third-party applications integrate with Symantec Protection Engine for a variety of content scanning services. You can configure client applications to communicate with Symantec Protection Engine using one of several supported protocols. However, the only protocol that is supported in the software developer's guide is the Internet Content Adaptation Protocol (ICAP) version 1.0, as presented in RFC 3507 (April 2003).

What's new in Symantec Protection Engine 8.1

Table 1-1 describes the new features in Symantec Protection Engine.

Table 1-1 New features

Feature	Description
Support for non-archive files larger than 2 GB	Symantec Protection Engine 8.1 supports the scanning of the non-archive files that are larger than 2 GB. The support is limited to 2 GB in previous releases.
Latest Symantec technologies	Symantec Protection Engine 8.1 is integrated with latest internal Symantec scanning technologies. Improved efficacy with the help of AML, x86 PE Emulator, nonPE/Scripts, etc.
Enhanced LiveUpdate	Internal critical fixes are now delivered through LiveUpdate.
Improved in-memory file system	Symantec Protection Engine uses the system memory to stream and scan the files. Now, the memory size is no more limited to 4 GB.

About integrating with Symantec Protection Engine

Symantec Protection Engine can be easily implemented in an existing infrastructure. Symantec Protection Engine runs on 64-bit platforms for Red Hat Linux and Windows Server. You can run Symantec Protection Engine on the same computer or a different computer than the client application

For more information about Symantec Protection Engine system requirements, see the *Symantec Protection Engine Implementation Guide*.

Software developers can create client applications (connectors) that let third-party applications integrate with Symantec Protection Engine for content scanning services using ICAP as the communication protocol.

You can create a custom integration using any of the following methods:

Construct an ICAP client connector using application program interface (API) C library.	If you plan to integrate antivirus scanning only, you can use the antivirus C API. URL filtering is not available using the antivirus API. The antivirus API includes static and dynamic libraries for each supported platform. The API library consists of functions that provide scanning and repair services to client applications. The C API supports the FILEMOD and RESPMOD scanning modes, and it contains the built-in ability to stream files. For more information, see Symantec_Protection_Engine_SDK\C\Docs\ in the Symantec Protection Engine documentation zip file.
Construct an ICAP client connector using Java API library.	If your environment has Java, you can use the Java API plug-in (SymJavaAPI.jar) to integrate with Symantec Protection Engine. The Java API provides client antivirus scanning and repair services using the ICAP protocol. The Java API supports the FILEMOD and RESPMOD scanning modes, and it contains the built-in ability to stream files. For more information, see Symantec_Protection_Engine_SDK\Java\Docs\SymJavaAPIDocs.jar in the Symantec Protection Engine documentation zip file.
Construct an ICAP client connector using .NET API library.	If your environment has .NET Framework, you can use the .NET API plug-in (symcsmsnetapi.dll) to integrate with Symantec Protection Engine. The .NET API provides client antivirus scanning and repair services using the ICAP protocol. The .NET API supports the FILEMOD and RESPMOD scanning modes, and it contains the built-in ability to stream files. For more information, see Symantec_Protection_Engine_SDK\CSharp\Docs\SymCSharpAPIDocs.chm in the Symantec Protection Engine documentation zip file.
Construct your own ICAP 1.0 client for the Symantec Protection Engine.	If you construct your own ICAP client, you can specify whether to perform antivirus scanning and URL filtering for outgoing and incoming requests. For more information, see *Symantec™ Protection Engine Software Developer's Guide*.

About ICAP

ICAP is a lightweight protocol that was originally created to execute a remote procedure call on HTTP messages. ICAP is part of an evolving architecture that lets corporations, data communication companies, and Internet service providers (ISPs) dynamically scan, change, and augment data as it flows through ICAP servers. The protocol lets ICAP clients pass data to ICAP servers for adaptation (some type of transformation or other processing, such as virus or URL filtering). The server executes its transformation service on the data and responds to the client, possibly with modified content.

In a typical integration for processing HTTP traffic, a caching proxy server retrieves the requested information from the Web. At the same time, it caches the information (stores a copy on disk) and, where possible, serves multiple requests for the same Web content from the cache. A caching proxy server can use ICAP to communicate with Symantec Protection Engine and request that content that is retrieved from the Web be scanned and repaired, if necessary.

About the C API

You can use the client-side API C library to configure an application to pass files to Symantec Protection Engine for scanning. The API includes static and dynamic libraries for each supported platform. The API library consists of functions that provide scanning and repair services to client applications.

Table 1-2 lists the available platforms for Symantec Protection Engine API C library

Table 1-2 Platforms for Symantec Protection Engine API C library

Operating system	Architecture	Compiler
Red Hat Enterprise Linux 5.5	x64	using gcc 4.1.2
Red Hat Enterprise Linux 5.5	x86	using gcc 4.1.2
Solaris 10 (SPARC)	64bit	using gcc 3.4.3
Windows Server 2008 R2	x64	using Microsoft Visual Studio 2008, 2010, or 2013
Windows Server 2008	x86	using Microsoft Visual Studio 2008

A C header file is included at the following location:

```
<Product .zip file>/Symantec_protection_engine_SDK/C/Include/
```

About other protocols

Symantec Protection Engine supports Remote Procedure Call (RPC) 1.0 and 1.1 for Network Appliance (NetApp) Filer. This protocol is a proprietary implementation and is not documented by Symantec.

Enhanced categorization of threats

Symantec Protection Engine categorizes each viral and non-viral threat under a top-level category (denoted as UberCategory). Each UberCategory has sub-categories associated with it, that are dynamic in nature and may change with definition updates. A threat can have only

one sub-category assigned to it. In addition, Symantec Protection Engine also provides sub-category ID and a short description about the sub-category.

Symantec measures each viral and non-viral threat using risk rating factors along with their impact levels. The risk rating factors and their impact levels defines the degree to which a threat can harm the computer.

Table 1-3 Risk rating factors and their impact levels

Factor	Description	Impact Level
Performance impact	Degree to which any program can affect a system's stability, speed, and performance.	■ High ■ Medium ■ Low
Privacy impact	Degree to which any program can gather personal information through the Internet and relay it back to a remote computer without the user's knowledge.	■ High ■ Medium ■ Low
Ease of removal	Degree to which any program can embed themselves deep within the system and refuse to be removed.	■ High ■ Medium ■ Low
Stealth	Degree to which any program can install itself without the user noticing and then remain hidden to prevent detection and removal..	■ High ■ Medium ■ Low

Symantec Protection Engine calculates the overall impact (Cumulative Risk Rating) of a detected threat based on the impact levels of individual risk rating factors. The detailed information for each detected threat is logged to all configured logging destinations and as part of the ICAP response.

Following is the sample uber category information in ICAP extension header X-Violations-Found when unrepairable virus Eicar is scanned:

EICAR Test String| ViralThreat=Virus| UberCategoryName=Malware| SubCategoryID=0| CumulativeRiskRating=High| PerformanceImpact=High| PrivacyImpact=High | EaseOfRemoval=High| Stealth=High| SubCategoryDescription= Programs that infect other programs, files, or areas of a computer by inserting themselves or attaching themselves to that medium

Enabling enhanced categorization of threats

To enable threat categorization in Symantec Protection Engine

1. Configure specific parameters in configuration.xml on the Symantec Protection Engine server.
   - EnableThreatCategoryInformation
   - EnableSubCategoryDescriptionResp

   For more information, see *Symantec Protection Engine Implementation Guide*.
2. Ensure that the following functions and problem attributes are used while configuring the client:

New functions	■ ScanClientScanFileEx() Used in file-based scanning. ■ ScanClientScanFileInsight() See "File-based scanning" on page 21. ■ ScanClientStreamStartEx() Used for stream-based scanning. ■ ScanClientStreamStartInsight() See "Stream-based scanning" on page 21. **Note:** These new API functions will use the new SYMCScanRespEx-AV service to send scan request to Symantec Protection Engine. For more information on services, see *Symantec Protection Engine Software Developer's Guide*.

New problem attributes

The nAttribute parameter in the ScanResultGetProblem function has the following new problem attributes:

- SC_PROBLEM_EX_FILENAME
- SC_PROBLEM_EX_VIOLATION_NAME
- SC_PROBLEM_EX_VIOLATION_ID
- SC_PROBLEM_EX_DISPOSITION
- SC_PROBLEM_EX_DEFINITION_DATE
- SC_PROBLEM_EX_DEFINITION_REV
- SC_PROBLEM_EX_NONVIRAL_THREAT_CATEGORY
- SC_PROBLEM_EX_FILE_UNSCANABLE
- SC_PROBLEM_EX_VIRAL_THREAT_CATEGORY
- SC_PROBLEM_EX_UBERCATEGORIES
- SC_PROBLEM_EX_SUBCATEGORY_ID
- SC_PROBLEM_EX_CUMULATIVE_RISK_RATING
- SC_PROBLEM_EX_PERFORMANCE_IMPACT
- SC_PROBLEM_EX_PRIVACY_IMPACT
- SC_PROBLEM_EX_EASE_OF_REMOVAL
- SC_PROBLEM_EX_STEALTH
- SC_PROBLEM_EX_SUBCATEGORY_DESCRIPTION
- SC_PROBLEM_EX_EXTRA_THREAT_INFO
- SC_PROBLEM_EX_INSIGHT_CACHE_DATE
- SC_PROBLEM_EX_INSIGHT_CACHE_REV
- SC_PROBLEM_EX_INSIGHT_SETTINGS_DATE
- SC_PROBLEM_EX_INSIGHT_SETTINGS_REV
- SC_PROBLEM_EX_INSIGHT_SYMVT_DATE
- SC_PROBLEM_EX_INSIGHT_SYMVT_REV
- SC_PROBLEM_EX_INSIGHT_RESULT
- SC_PROBLEM_EX_REPUTATION
- SC_PROBLEM_EX_AGE
- SC_PROBLEM_EX_PREVALENCE
- SC_PROBLEM_EX_FILETYPE
- SC_PROBLEM_EX_SHA256
- SC_PROBLEM_EX_EXTRA_INSIGHT_INFO
- SC_PROBLEM_EX_EXTRA_APK_ RESULT_INFO
- SC_PROBLEM_EX_APK_RESULT_ID
- SC_PROBLEM_EX_INTERNAL_SERVER_ERR_REASON

See "ScanResultGetProblem parameters" on page 37.

About backward compatibility

If you choose to configure your C API client with older API function calls (ScanClientScanFile() and ScanClientStreamStart()) and old problem attributes, the C API version 7.x will not return the enhanced threat categorization information.

This would mean that 7.x C SDK users can continue to send scanning requests to older versions of Symantec Scan Engine with no compatibility issues.

Better handling of Unscannable files

Symantec Protection Engine terms encrypted and malformed containers as "Unscannable" files. Previously, Symantec Scan Engine had limited policies to handle such unscannable files. Now, Symantec Protection Engine provides better control over handling unscannable files.

Symantec Protection Engine provides the following options to handle unscannable files:

- Log only
  The file is treated as Clean, and generates a log entry
- Block
  Blocks the unscannable files and generates a log entry
- Delete
  Deletes the unscannable files and generates a log entry

For more information, see *Symantec Protection Engine Implementation Guide*.

To identify unscannable files, the following two new problem attributes have been introduced:

- SC_PROBLEM_FILE_UNSCANABLE (applicable to non-threat categorization API function calls, for example, ScanClientScanFile() and ScanClientStreamStart())
- SC_PROBLEM_EX_FILE_UNSCANABLE (applicable to threat categorization API function calls, for example, ScanClientScanFileEx() and ScanClientStreamStartEx())

These new problem attributes have been introduced in the nAttribute parameter of ScanResultGetProblem() function. The attribute will return true if the file is Unscannable, else it will return false.

See “ScanResultGetProblem parameters” on page 37.

About Keep-Alive connection support

Previously, the C API library only supported creating a new connection for every single request/response pair. If the average file size is small or if scan requests are frequent (say, during peak hours), creating and closing a new connection for each scan request is memory and CPU intensive as it involves a lot of handshaking subsequently increasing network congestion.

Now, from version 7.0 onwards, the C API library additionally supports ICAP persistent connections, also known as ICAP keep-alive, which uses the same TCP connection to send and receive multiple ICAP requests/responses. When the client sends another request, it uses the same connection. This will continue until either the client or the server drops the connection, or, the connection times out due to no activity on the connection.

The keep-alive connection reduces frequent opening and closing connections, resulting in lesser CPU/memory usage, lesser network congestion and reduced latency in subsequent requests when compared to the legacy method.

The client application provides the maximum required persistent connection pool size to the server during initialization. Depending on the scan request load, the number of persistent connections will dynamically increase to the specified maximum number. If all connections are busy and no more connections can be created (more than the specified keep-alive connection), and the client tries to send a scan request, the API will return an error (SCSCANFILE_CONNPOOLSIZE_EXCEEDED or SC_CONNPOOLSIZE_EXCEEDED). In case the client does not send scanning requests or persistent connections remain idle for more than 5 minutes, Symantec Protection Engine will close the connection and free up the server resources.

To implement the keep-alive connection method, you must meet the following guidelines:

- Client connector must have multi-threaded support.
- Configure the maximum number of threads on Symantec Protection Engine to be more than or equal to the total number of keep- alive connections configured on the client(s). For more information on how to configure the “MaxThreads” see, *Symantec Protection Engine Implementation Guide*.
- To ensure that all scan requests are served immediately upon arrival, configure the minimum number of threads on Symantec Protection Engine (MinThreads) to be equal to the number of persistent connections (NumberOfKeepAliveConnections). This is required to handle scenarios where there is a sudden increase in scan requests, as there is a time lag for increasing the number of available threads from the configured minimum number.
  For more information on how to configure the “MinThreads” see, *Symantec Protection Engine Implementation Guide*.

Chapter 2

Constructing clients using the antivirus client-side API library

This chapter includes the following topics:

- General procedure for scanning
- Compiling and linking
- API functions

General procedure for scanning

The Symantec Protection Engine client API library provides a set of functions to simplify communication with Symantec Protection Engine using ICAP.

The procedure for scanning data is as follows:

- The client calls ScanClientStartUp() once to initialize the client library and to set up the scheduling.
- Files are scanned using either ScanClientScanFile() or ScanClientStreamStart(), ScanClientStreamSendBytes() (as many times as necessary), and ScanClientStreamFinish().

Note: For enhanced threat categorization, use either ScanClientScanFileEx(), ScanClientStreamStartEx(), ScanClientScanFileInsight(), or ScanClientStreamStartInsight() instead of ScanClientScanFile() or ScanClientStreamStart().

- Information on infections found (if any) is obtained using ScanResultGetNumProblems() and ScanResultGetProblem().
- Completion of scanning for a particular file is indicated using ScanResultsFree().
- Completion of all scanning is indicated (for example, before program termination) using ScanClientShutDown() to free resources.

Sample code is included in the *Symantec Protection Engine Software Developer's Guide* for convenience. The sample code is also included in the distribution package. The sample program demonstrates how to use Symantec Protection Engine API 192.168.1.2 to scan files.

Compiling and linking

Table 2-1 describes requirements for compiling and linking on Solaris, Red Hat Linux, and Windows platforms

Table 2-1 Compiling and linking requirements

Platform	Include	Initialize	Link
Solaris SPARC x64	#include "symcsapi.h"	none	libsocket.a libnsl.a (for example, gcc -lsocket -lnsl)
Red Hat Linux x86/x64	#include "symcsapi.h"	none	libnsl.a (for example, gcc -lnsl)
Windows Server 2008/Windows Server 2008 R2 x64	#include "symcsapi.h"	initialize winsock	winsock (for example, WS2_32.LIB)

Note: The code (in all versions of all supported platforms) is compiled with position-independent code so that it can be used in shared libraries.

Note: For gcc 3.x, you must link to the Standard C++ library (`-lstdc++`).

Compiling on Windows Server 2008 R2 x64

To compile on Microsoft Windows Server 2008 R2 x64, use Microsoft Visual Studio 2008 (version 9.0), Microsoft Visual Studio 2010 (version 10.0), or Microsoft Visual Studio 2013 (version 12.0)

The source code should include the symcsapi.h file and link to symcsapi.lib. The application should be compiled with multi-threaded runtime libraries. For example, in Microsoft Visual

Studio, under Project Settings, click the C/C++ tab, click Code Generation, and click Multithreaded Libraries.

Note: You must link to the winsock library (WS2_32.LIB) and initialize winsock in the source code before scanning files. You must release winsock when all network access is complete (usually just before exiting the program).

Initializing winsock

The client application must initialize winsock before scanning files. The following example demonstrates winsock initialization:

```
#include <windows.h>
.
.
.
// start up winsock
WORD wVersionRequested;
WSADATA wsaData;
  int err;

// Load WinSock, request version 1.0.
wVersionRequested = MAKEWORD(1, 0);
err = WSAStartup(wVersionRequested, &wsaData);
if (err != 0)
{
// ERROR
  }

// Confirm WinSock supports the version we requested.
if (LOBYTE(wsaData.wVersion) != 1 ||
HIBYTE(wsaData.wVersion) != 0)
{
WSACleanup();
// ERROR
  }
```

Shutting down winsock

You must release winsock when all network access is complete.

```
if (WSACleanup() == SOCKET_ERROR)
     ;// ERROR
```

Solaris

Table 2-2 describes the different versions of gcc compilers for different version of Solaris.

Table 2-2 Compilers for different versions of Solaris

Solaris version	gcc compiler version
Solaris 10 (SPARC) -64bit	3.4.3

The source code that makes calls to the library should include the symcsapi.h header file. In the makefile (or command line) that compiles the program, libsymcsapi.a should be added and, if it is not already used, -lnsl -lsocket should be added. For example:

gcc mycode.c libsymcsapi.a -lsocket -lnsl

On the x64 platform, a new flag -m64 should be added. For example:

gcc -m64 mycode.c libsymcsapi.a -lsocket –lnsl

Note: If you are compiling your own multithreaded application, you must define the reentrant symbol (-D_REENTRANT) so that multithreading functions properly. The API libraries are already compiled in this manner.

Red Hat Linux

Table 2-3 describes the different versions of gcc compilers for different versions of Linux.

Table 2-3 Compilers for different versions of Linux

Linux version	gcc compiler version
Red Hat Enterprise Linux 5.5 and later (x64)	4.1.2
Red Hat Enterprise Linux 5.5 and later (x86)	4.1.2

The source code that makes calls to the library should include the symcsapi.h header file. In the makefile (or command line) that compiles the program, libsymcsapi.a should be added and, if it is not already used, -lnsl should be added. For example:

gcc mycode.c libsymcsapi.a -lnsl

On the x64 platform, a new flag -m64 should be added. For example:

gcc -m64 mycode.c libsymcsapi.a -lnsl

Note: If you are compiling your own multithreaded application, you must define the reentrant symbol (-D_REENTRANT) so that multithreading functions properly. The API libraries are already compiled in this manner.

Exceptions and error handling

The Symantec Protection Engine API C library does not throw exceptions or include exception handling. Detected errors are returned as result codes from the functions.

API functions

The API functions are as follows:

- ScanClientStartUp
- ScanClientSecureStartUp
- ScanClientScanFile
- ScanClientScanFileEx
- ScanClientScanFileInsight
- ScanResultGetNumProblems
- ScanResultGetProblem
- SC_DECODE_DISPOSITION
- ScanResultsFree
- ScanClientShutDown
- ScanClientSecureShutDown
- ScanClientStreamStart
- ScanClientStreamStartEx
- ScanClientStreamStartInsight
- ScanClientStreamSendBytes
- ScanClientStreamFinish
- ScanClientStreamAbort
- ScanGetNumConnectErrors
- ScanGetConnectError

File-based scanning

If you are developing a file-based-scanning client, you should use the following functions:

- ScanClientStartUp()
- ScanClientScanFile()
  -or-
  ScanClientScanFileEx()
  -or-
  ScanClientScanFileInsight
- ScanResultGetNumProblems()
- ScanResultGetProblem()
- ScanResultsFree()
- ScanClientShutDown()

Stream-based scanning

If you are developing a stream-based-scanning client, you should use the following functions:

- ScanClientStartUp()
- ScanClientStreamStart()
  -or-
  ScanClientStreamStartEx()
  -or-
  ScanClientStreamStartInsight
- ScanClientStreamSendBytes()
- ScanClientStreamFinish()
- ScanClientStreamAbort()
- ScanResultGetNumProblems()
- ScanResultGetProblem()
- ScanResultsFree()
- ScanClientShutDown()

ScanClientStartUp

The ScanClientStartUp function lets a new client begin submitting files to Symantec Protection Engine.

```
SC_ERROR ScanClientStartUp
(
HSCANCLIENT*client,
LPSTRpszClientConfiguration
)
```

ScanClientStartUp parameters

Table 2-4 lists the parameters that are used.

Table 2-4 ScanClientStartUp parameters

Parameter	Description
client	Address of an HSCANCLIENT variable. The HSCANCLIENT variable is a handle to a status data structure used throughout the scanning process. This variable contains information about the servers that are being used and the scheduling mechanism. Memory is allocated for this data structure by this call and must be freed using ScanClientShutDown() when scanning is completed.

Table 2-4 ScanClientStartUp parameters *(continued)*

Parameter	Description
pszClientConfiguration	A null-terminated string that contains configuration information. Entries are separated with three semicolons (;;;). No spaces are allowed. The following options are supported: ■ Server: ipaddress:port (legacy format) All Symantec Protection Engines that are used should be listed. At least one server must be listed. ■ Server:ipaddress:port:NumberOfKeepAliveConnections(keep-alive format) where NumberOfKeepAliveConnections is per Symantec Protection Engine server. All Symantec Protection Engines that are used should be listed. At least one server must be listed. **Note:** For legacy behavior specifying port is optional, but for keep-alive (persistent) connection feature, port is mandatory. ■ FailRetryTime:<seconds> If the client fails to connect to a Symantec Protection Engine, wait <seconds> before trying to connect to that server again (use only the other servers in the meantime, unless they have all failed recently). The default setting is 30 seconds. ■ ReadWriteTime:<seconds> If after <seconds> no response is received from Symantec Protection Engine (when data is being transmitted to Symantec Protection Engine), or if the transmission of data does not complete (when data is being receiving from Symantec Protection Engine), return an error message. The default setting is 30 seconds.

ScanClientStartUp return codes

Table 2-5 lists the return codes (negative values are warnings; positive values are errors).

Table 2-5 ScanClientStartUp return codes

Value	Return code	Description
3	SC_MEMORY_ERROR	A memory allocation error has occurred.
1	SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
0	SC_OK	Success.

ScanClientStartUp description

In a multithreaded environment, the client should make a single call to ScanClientStartUp on behalf of all of its threads. This way, all threads use scheduling through a single scheduler rather than through multiple schedulers.

Parameter pszClientConfiguration should look like the following:

Server:1.2.3.4:1344;;;Server:1.2.3.5:1344

At least one Symantec Protection Engine must be specified. Entries are separated with three semicolons. No spaces are allowed. If the default port (1344) is used, the port number (and the colon) can be omitted.

In case of persistent (keep-alive) connections, the parameter pszClientConfiguration should look like the following:

Server:1.2.3.4:1344:16;;;Server:1.2.3.5:1344:8

In this example, the API will maintain 16 and 8 connections with the Symantec Protection Engine server 1.2.3.4 and 1.2.3.5 respectively.

Note: When multiple protection engines are specified, the API provides automatic scheduling across any number of protection engines. The API determines the appropriate protection engine to receive the next file to be scanned, based on the scheduling algorithm. If a protection engine is unreachable or stops responding during a scan, another protection engine is called and the faulty protection engine is taken out of rotation for a period of time (30 seconds is the default setting). If all of the Symantec Protection Engines are out of rotation, the faulty protection engines are called again. The API does not stop trying to contact the Symantec Protection Engines unless five engines do not respond or it appears that a file that is being scanned might have caused more than one engine to stop responding.

ScanClientSecureStartUp

```
The ScanClientSecureStartUp function lets you configure Symantec Protection Engine ser

SC_ERROR DECLSPEC_SYMCSAPI ScanClientSecureStartUp
( HSCANCLIENT *client,
 LPSTR pszClientConfiguration,
 SecureConnectionInfo* secureConInfo);
```

ScanClientSecureStartUp parameters

Table 2-6 lists the parameters that are used.

Table 2-6 ScanClientSecureStartUp parameters

Parameters	Description
client	Address of an HSCANCLIENT variable. The HSCANCLIENT variable is a handle to a status data structure used throughout the scanning process. This variable contains information about the servers that are being used and the scheduling mechanism. Memory is allocated for this data structure by this call and must be freed using ScanClientSecureShutDown() when scanning is completed.

Table 2-6 ScanClientSecureStartUp parameters *(continued)*

Parameters	Description
pszClientConfiguration	A null-terminated string that contains configuration information. Entries are separated with three semicolons (;;;). No spaces are allowed. The following options are supported: ■ Server Following are the three supported formats for configuring Server information in this API. ■ server:ipaddress:port:NumberOfKeepAliveConnections:<true/false> List all the Symantec Protection Engines that are used. At least one server must be listed. Where, NumberOfKeepAliveConnections is per Symantec Protection Engine server. **true** indicates that secure ICAP is enabled at server and communication should be done using secure ICAP. **false** indicates that secure ICAP is not enabled and communication should be done using plain ICAP. For secure ICAP communication, you must specify port and NumberOfKeepAliveConnections. If persistent connection feature is not required with secure ICAP, pass 0 as NumberOfKeepAliveConnections in Server information, that is, "ipaddress:port:0:<true/false>" ■ Server:ipaddress:port:NumberOfKeepAliveConnections(keep-alive format) Where, NumberOfKeepAliveConnections is per Symantec Protection Engine server. List all the Symantec Protection Engines that are used. At least one server must be listed. **Note:** For legacy behavior, specifying port is optional, but for keep-alive (persistent) connection feature, port is required. ■ Server: ipaddress:port (legacy format) List all the Symantec Protection Engines that are used. At least one server must be listed. ■ FailRetryTime:<seconds> If the client fails to connect to a Symantec Protection Engine, wait <seconds> before trying to connect to that server again (use only the other servers in the meantime, unless they have all failed recently). The default setting is 30 seconds. ■ ReadWriteTime:<seconds> If after <seconds> no response is received from Symantec Protection Engine (when data is being transmitted to Symantec Protection Engine), or if the transmission of data does not complete (when data is being receiving from Symantec Protection Engine), return an error message. The default setting is 30 seconds.

Table 2-6 ScanClientSecureStartUp parameters *(continued)*

Parameters	Description

SecureConnectionInfo	You must fill certificate file specific information to use secure ICAP feature. If you have no any specific information, initialize pointers with NULL. In that case, this API will only be able to connect to server with plain ICAP.
	SecureConnectionInfo structure:
	`typedef struct SecureConnectionInfo`
	`{`
	`char* CACertFile;`
	`char* CACertFilePath;`
	`char* ClientCertificateFile;`
	`char* ClientPrivateKeyFile;`
	`char* ClientPrivateKeyPassword;`
	`char* CipherList;`
	`bool* VerifyServerCert;`
	`} SecureConnectionInfo;`
	■ CACertFile must be provided for using secure ICAP feature when VerifyServerCert option is set to true.
	VerifyServerCert option is true by default.
	■ CACertFilePath is an optional parameter. It must point to a directory containing CA certificates in PEM format, or must be initialized with NULL
	■ ClientCertificateFile, ClientPrivateKeyFile, and ClientPrivateKeyPassword is required only if client authentication is on at server, otherwise must be set to NULL.
	■ CipherList values must be separated by comma(,) or colon (:). for example,
	"AES256-SHA256,AES128-SHA,AES256-SHA,DES-CBC3-SHA"

ScanClientSecureStartUp return codes

Table 2-7 lists the return codes (negative values are warnings; positive values are errors).

Table 2-7 ScanClientSecureStartUp return codes

Value	Return code	Description
1	SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
3	SC_MEMORY_ERROR	A memory allocation error has occurred.

Table 2-7 ScanClientSecureStartUp return codes *(continued)*

Value	Return code	Description
9	SC_SECURE_INITIALIZATION_FAILURE	Failed to initialize SSL/TLS library, typically at the time of application start.
0	SC_OK	Success.

ScanClientSecureStartUp description

In a multithreaded environment, the client should make a single call to ScanClientSecureStartUp on behalf of all its threads. This way, all threads use scheduling through a single scheduler rather than through multiple schedulers.

In case of secure ICAP configuration, the parameter pszClientConfiguration should look like the following:

- server:ipaddress:port:NumberOfKeepAliveConnections:<true/false>;;; FailRetryTime:30;;;ReadWriteTime:30 (Secure ICAP format)
- server:ipaddress:port:NumberOfKeepAliveConnections:<true/false>;;;server:ipaddress:port: NumberOfKeepAliveConnections:<true/false>;;;FailRetryTime:30;;;ReadWriteTime:30 (Secure ICAP format if there are 2 servers)

At least one Symantec Protection Engine must be specified. Entries are separated with three semicolons. No spaces are allowed.

true indicates that secure ICAP is enabled at server and communication should be done using secure ICAP. false indicates that secure ICAP is not enabled and communication should be done using plain ICAP.

For secure ICAP communication, specifying port and NumberOfKeepAliveConnections is must. If persistent connection feature is not required with secure ICAP, just pass 0 as NumberOfKeepAliveConnections in Server information, that is, "ipaddress:port:0:<true/false>"

ScanClientScanFile

The ScanClientScanFile function is called to have Symantec Protection Engine scan a file for viruses.

Note: The OriginalFileName parameter must be Unicode UTF-8 encoded.

```
SCSCANFILE_RESULT DECLSPEC_SYMCSAPI ScanClientScanFile
(
 HSCANCLIENT   hScanClient,
 LPSTR         pszOriginalFileName,
```

```
 LPSTR           pszActualFileName,
 LPSTR           pszOutputFileName,
 LPSTR           pszConfigPolicy,
 LPHSCANRESULTS  phScanResults
)
```

ScanClientScanFileEx

The ScanClientScanFile and ScanClientScanFileEx function have same parameters.

```
SCSCANFILE_RESULT DECLSPEC_SYMCSAPI ScanClientScanFileEx
(
 HSCANCLIENT    hScanClient,
 LPSTR          pszOriginalFileName,
 LPSTR          pszActualFileName,
 LPSTR          pszOutputFileName,
 LPSTR          pszConfigPolicy,
 LPHSCANRESULTS phScanResults
);
```

ScanClientScanFileInsight

The ScanClientScanFileInsight has the following parameters:

```
SCSCANFILE_RESULT DECLSPEC_SYMCSAPI ScanClientScanFileInsight
(
 HSCANCLIENT      hScanClient,
 LPSTR            pszOriginalFileName,
 LPSTR            pszActualFileName,
 LPSTR            pszOutputFileName,
 LPSTR            pszConfigPolicy,
 int              disableInsightCall,
 InsightOptions*  insightOptions,
 LPHSCANRESULTS   phScanResults
);
```

ScanClientScanFile parameters

Table 2-8 lists the parameters that are used.

Table 2-8 ScanClientScanFile parameters

Parameter	Description
hScanClient	An HSCANCLIENT variable that has been initialized. See "ScanClientStartUp" on page 21.
pszOriginalFileName	The name of the file to be scanned. (The original name of the file on the user's computer.) This parameter is ignored if the PassFileByName=1 string is set in pszConfigPolicy.
pszActualFileName	The name (path) of the file to scan on the client's computer. This path might be different than pszOriginalFileName. For example, a user may upload a file to be scanned called sample.doc, but the same file may be stored on the ISP computer as temp123.doc. In this case, the pszOriginalFileName is sample.doc, but the pszActualFileName may be /tmp/temp123.doc.
pszOutputFileName	The storage location for the repaired file. (No output file is created unless the input file is infected and the infection is repairable.) This parameter can be any of the following: ■ A null-terminated string that is a path to where the repaired file is to be stored. ■ A char array at least MAX_STRING long with the first byte set to '\0'. The API generates a file name for the repair file. When the function returns, pszOutputFileName is set to the name of the repaired file. **Note:** If this parameter is NULL, the API has Symantec Protection Engine scan the file for viruses but not attempt repair. However, this is not the recommended method for forcing Symantec Protection Engine to scan files for viruses but not attempt repair. You should set the pszConfigPolicy to ScanOnly instead. If local scanning options are set using pszConfigPolicy, Symantec Protection Engine ignores this parameter and repairs the file in place.

Table 2-8 ScanClientScanFile parameters *(continued)*

Parameter	Description
pszConfigPolicy	A null-terminated string of the form <option:value>;;;<option:value>… No spaces are allowed. The following options are supported: ■ ScanOnly:1: Scan for viruses but do not attempt repair. ■ AlwaysReportDefInfo:1: If a clean file is scanned, a Problem Incident is created with only the virus definitions date and revision number. ■ RepairOnly:1: Attempt to repair infected files but do not delete the files that cannot be repaired. ■ ICAPClientIP:<IP address>: Specify IP source address of the encapsulated HTTP request. The IP address must be a dotted-decimal IPv4 address. It is used in ICAP request header X-Client-IP. ■ PassFileByName:1: The file to be scanned is on the same computer as Symantec Protection Engine or can be accessed by network file system (NFS) or another network file protocol. In this case, the file is not sent over a socket. Instead, Symantec Protection Engine opens the file directly for scanning. If a repair is required, the repair is made in the local directory. If this string is set, Symantec Protection Engine uses the pszActualFileName parameter to read the file and uses the ExtensionList= and ExclusionList= options to determine how to handle scanning for a specific file type. This functionality is not available for this release. ■ ReportInsightInfo: ■ 0: Symantec Protection Engine does not provide reputation information in ICAP response. ■ 1: Symantec Protection Engine provides reputation information in ICAP response for Insight convicted files. **Note:** If options scanonly or repaironly are not set, then scanrepairdelete is used as the default scanning policy.
phScanResults	When the function returns, this handle points at a structure that contains information about any problems (infections) found during the scan. If none were found, this parameter is NULL unless the AlwaysReportDefInfo:1 policy is used in the scan. Information about problems is extracted using the described functions. If this parameter is not NULL, the memory must be released using ScanResultsFree().
disableInsightCall	This parameter is used only for ScanClientScanFileInsight
InsightOptions*	This parameter is used only for ScanClientScanFileInsight

Table 2-8 ScanClientScanFile parameters *(continued)*

Parameter	Description

insightOptions	Structure of input for insight scanning
	
	`typedef struct InsightOptions`
	`{`
	`int insightAggressionLevel;`
	`char* MD5Hash;`
	`char* SHA256Hash;`
	`int isDigitallySigned;`
	`char* fileSourceURL;`
	`char* fileSourceIP;`
	`}InsightOptions;`
	
	This parameter is used only for ScanClientScanFileInsight

ScanClientScanFile return codes

Table 2-9 lists the return codes for ScanClientScanFile (negative values indicate that a virus was detected; positive values are errors; zero indicates a clean file).

Table 2-9 ScanClientScanFile return codes

Value	Return code	Description
-3	SCSCANFILE_INF_NO_REP	The file was infected with a virus, but no repair was possible.
-1	SCSCANFILE_INF_REPAIRED	The file was infected, and repair was successful.
0	SCSCANFILE_CLEAN	No virus was found.
1	SCSCANFILE_FAIL_CONNECT	An attempt to connect to a Symantec Protection Engine failed.
2	SCSCANFILE_FAIL_INPUTFILE	A problem was encountered reading the file to be scanned.
3	SCSCANFILE_FAIL_ABORTED	The scan was aborted abnormally.
4	SCSCANFILE_INVALID_PARAM	A function was called with an invalid parameter.
5	SCSCANFILE_FAIL_RECV_FILE	An error occurred when attempting to receive the repaired file.
6	SCSCANFILE_FAIL_MEMORY	A memory allocation error occurred.

Table 2-9 ScanClientScanFile return codes *(continued)*

Value	Return code	Description
7	SCSCANFILE_FAIL_FILE_ACCESS	The server could not access the file to be scanned. This error usually occurs for LOCAL scans when the file permissions are wrong or when the file is not in the path that is specified in the LocalFileScanDir parameter on the server. This error can also occur when the API encounters a problem while writing repaired file data that was received from Symantec Protection Engine to the output file.
10	SCSCANFILE_ERROR_SCANNINGFILE	An internal server error occurred while Symantec Protection Engine was attempting to repair the file.
15	SCSCANFILE_ABORT_NO_AV_SCANNING_LICENSE	No valid license for antivirus scanning functionality is installed.
17	SCSCANFILE_FAIL_SERVER_TOO_BUSY	The Symantec Protection Engine server is busy and unable to process the scan request.
18	SCSCANFILE_CONNPOOLSIZE_EXCEEDED	Maximum persistent connection limit has been reached. No more connections allowed.

ScanClientScanFile description

The ScanClientScanFile function determines the appropriate Symantec Protection Engine (when multiple protection engines are running) based on the scheduling algorithm. If a protection engine is unreachable or goes down during a scan, another server is called and the faulty server is taken out of rotation for a period of time. If all protection engines are out of rotation, the faulty servers are called again. The ScanClientScanFile function does not stop trying to contact a protection engine unless five servers are not functioning or it appears that a file that is being scanned might have caused two servers to go down.

ScanClientScanFileEx description

If the ScanClientScanFiletEx() function is used (instead of ScanClientScanFile()), then an extended problem attribute set will be available from Symantec Protection Engine, provided the required parameters are enabled on the server. This extended problem attribute set contains enhanced categorization information about the detected threat.

The ScanClientScanFile() and ScanClientScanFileEx() function have same parameters and return codes.

ScanClientScanFileInsight description

If the ScanClientScanFileInsight() function is used, then Symantec Protection Engine will additionally query Symantec Reputation servers to get the additional reputation result for specified file. The ScanClientScanFileInsight() function has 2 additional optional parameters compared to ScanClientScanFile() and ScanClientScanFileEx() functions.

disableInsightCall specifies whether to enable or disable insight server query. By default disableInsight is 0.

Attributes configured in InsightOptions structure can be used to query the reputation server for file reputation result. Following are the attributes :

- insightAggressionLevel - Insight aggression level used by SPE to convict file using reputation result.
- MD5Hash – MD5 hash of file
- SHA256Hash – SHA256 Hash of file.
- fileSourceIP – Source IP of the file.
- fileSourceURL- Source URL of the file.
- IsDigitallySigned – whether the fie being sent for scanning is digitally signed.

The return codes for ScanClientScanFileInsight() function are same as ScanClientScanFile() and ScanClientScanFileEx() functions.

ScanResultGetNumProblems

The ScanResultGetNumProblems function indicates the number of infections that are contained in an HSCANRESULT after scanning a file. Use the ScanResultGetProblem() function to get information about the infections that are reported in an HSCANRESULT after scanning a file.

```
SC_ERROR ScanResultGetNumProblems
(
HSCANRESULT hScanResult,
int *nNumProblems
)
```

ScanResultGetNumProblems parameters

Table 2-10 lists the parameters that are used.

Table 2-10 ScanResultGetNumProblems parameters

Parameter	Description
hScanResult	An HSCANRESULT variable that is returned by ScanClientScanFile(), ScanClientScanFileEx(), ScanClientStreamFinish(), ScanClientScanFileInsight(), or ScanClientStreamStartInsight().
nNumProblems	When the function returns, this parameter is set to the number of infections in the scanned file. If the AlwaysReportDefInfo:1 scan policy option is used, at least one (blank) incident is reported, along with the virus definitions date and revision number.

ScanResultGetNumProblems return codes

Table 2-11 lists the return codes (negative values are warnings; positive values are errors).

Table 2-11 ScanResultGetNumProblems return codes

Value	Return code	Description
0	SC_OK	Success.
-1	SC_NULL_PARAMETER	A parameter that was passed to the function is NULL when it should not be.

ScanResultGetNumProblemsEx

The ScanResultGetNumProblemsEx function gives additional information about the data scanned and the total files scanned in a scan request.

```
SC_ERROR DECLSPEC_SYMCSAPI ScanResultGetNumProblemsEx
(
                HSCANRESULT hScanResult,
                int *nNumProblems,
                LLINT     *DataScannedBytes,
                LLINT     *TotalFilesScanned
);
```

This function should be used only with the following:

- ScanClientScanFileEx
- ScanClientStreamStartEx
- ScanClientScanFileInsight
- ScanClientStreamStartInsight

To run this function, you must add the following parameters to the `category3.xml` file with the value as `True`.

Table 2-12 Parameters to be added in `category3.xml`

Name	XPath	Value
EnableDataScanned InICAPResponse	`/configuration/ProtocolSettings/ICAP/EnableDataScannedInICAPResponse/@value`	`True`
EnableTotalFiles ScannedInICAPResponse	`/configuration/ProtocolSettings/ICAP/EnableTotalFilesScannedInICAPResponse/@value`	`True`

ScanResultGetNumProblemsEx parameters

Table 2-13 lists the parameters that are used.

Table 2-13 ScanResultGetNumProblemsEx parameters

Parameter	Description
hScanResult	An *HSCANRESULT* variable that is returned by ScanClientScanFile(), ScanClientScanFileEx(), ScanClientStreamFinish(), ScanClientScanFileInsight(), or ScanClientStreamStartInsight().
nNumProblems	When the function returns, this parameter is set to the number of infections in the scanned file. If the AlwaysReportDefInfo:1 scan policy option is used, at least one (blank) incident is reported, along with the virus definitions date and revision number.
DataScannedBytes	This parameter returns the amount of data in bytes that is scanned by the scan engine.
TotalFilesScanned	This parameter returns the total number of files that are scanned. In the case of a container file such as a zip or an archive, the total number of files scanned includes the container file and the files contained within it. For example, if `file.zip` contains 4 files, then the total number of files scanned will be 5.

ScanResultGetNumProblemsEx return codes

Table 2-14 lists the ScanResultGetNumProblemsEx return codes (negative values are warnings; positive values are errors).

Table 2-14 ScanResultGetNumProblemsEx return codes

Value	Return code	Description
0	SC_OK	Success.
-1	SC_NULL_PARAMETER	The parameters that were passed to the function do not exist.

About ICAP response header

If you are using SDK, then the ICAP response status is followed by response headers. For more information about ICAP and response codes, refer the section *About ICAP responses* in the *Symantec™ Protection Engine 7.9 Software Developer's Guide* for NAS and Cloud Services.

Table 2-15 Response header

Header	Description
X-Data-Scanned-Bytes	This parameter returns the amount of data in bytes that is scanned by the scan engine.
X-Total-Files-Scanned	This parameter returns the total number of files that are scanned. In the case of a container file such as a zip or an archive, the total number of files scanned includes the container file and the files contained within it. For example, if `file.zip` contains 4 files, then the total number of files scanned will be 5.

ScanResultGetProblem

The ScanResultGetProblem function is used to get specific information about a virus.

```
SC_ERROR ScanResultGetProblem
(
HSCANRESULT hScanResult,
int nProblemNum,
int nAttribute,
LPSTR pszValueOut,
LPINT pnValueLengthInOut
)
```

ScanResultGetProblem parameters

Table 2-16 lists the parameters that are used.

Table 2-16 ScanResultGetProblem parameters

Parameter	Description
hScanResult	An HSCANRESULT variable that is returned by ScanClientScanFile(), ScanClientScanFileEx(), ScanClientStreamFinish(), ScanClientScanFileInsight(), or ScanClientStreamStartInsight().
nProblemNum	Integer specifying which problem entry is being investigated.

Table 2-16 ScanResultGetProblem parameters *(continued)*

Parameter	Description
nAttribute	

Table 2-16 ScanResultGetProblem parameters *(continued)*

Parameter	Description
	Identifies which attribute is being queried. If your client connector uses either any file-based scanning or stream-based scanning functions, then the valid attributes are as follows: ■ SC_PROBLEM_FILENAME The file name in which the infection occurred. ■ SC_PROBLEM_VIRUSNAME The name of the virus that has infected the file. In the case of violation, the value of this attribute returns the violation name. ■ SC_PROBLEM_NONVIRAL_THREAT_CATEGORY The name of the non-viral threat category. This attribute is sent only if you configure Symantec Protection Engine to send it. For more information, see the *Symantec Protection Engine Implementation Guide*. ■ SC_PROBLEM_VIRUSID The unique numerical ID of the virus. In the case of violation, the value of this attribute returns the violation ID. ■ SC_PROBLEM_DISPOSITION Problem resolution. This value is a number (in a string format) that indicates whether the virus was cleaned from the file. ■ SC_PROBLEM_DEFINITION_DATE The date stamp on the virus definitions. ■ SC_PROBLEM_DEFINITION_REV Revision number of the definitions. ■ SC_PROBLEM_FILE_UNSCANABLE In the case of unscannable file, the value of this attribute returns true else false. If your client connector uses the following API functions: **File-based scanning** ■ ScanClientScanFileEx() ■ ScanClientScanFileInsight() **Stream-based scanning** ■ ScanClientStreamStartEx() ■ ScanClientStreamStartInsight() Then the additional valid attributes are as follows: ■ SC_PROBLEM_EX_FILENAME The file name in which the infection occurred. ■ SC_PROBLEM_EX_VIOLATION_NAME The name of the virus that has infected the file. In the case of violation, the

Table 2-16 ScanResultGetProblem parameters *(continued)*

Parameter	Description
	value of this attribute returns the violation name. ■ SC_PROBLEM_EX_VIOLATION_ID The unique numerical ID of the virus. In the case of violation, the value of this attribute returns the violation ID. ■ SC_PROBLEM_EX_DISPOSITION Problem resolution. This value is a number (in a string format) that indicates whether the virus was cleaned from the file. ■ SC_PROBLEM_EX_DEFINITION_DATE The date stamp on the virus definitions. ■ SC_PROBLEM_EX_DEFINITION_REV Revision number of the definitions. ■ SC_PROBLEM_EX_NONVIRAL_THREAT_CATEGORY This parameter provides non-viral threat category under which a non-viral threat is been categorized in ICAP response.

Table 2-16 ScanResultGetProblem parameters *(continued)*

Parameter	Description

Table 2-16 ScanResultGetProblem parameters *(continued)*

Parameter	Description
	Continued... ■ SC_PROBLEM_EX_FILE_UNSCANABLE In the case of unscannable file, the value of this attribute returns true else false. ■ SC_PROBLEM_EX_VIRAL_THREAT_CATEGORY This parameter provides viral threat category under which a non-viral threat is been categorized in ICAP response. ■ SC_PROBLEM_EX_UBERCATEGORIES ■ SC_PROBLEM_EX_SUBCATEGORY_ID ■ SC_PROBLEM_EX_CUMULATIVE_RISK_RATING This parameter provides cumulative risk rating of threat. ■ SC_PROBLEM_EX_PERFORMANCE_IMPACT This parameter provides performance impact of threat. ■ SC_PROBLEM_EX_PRIVACY_IMPACT This parameter provides privacy impact of threat. ■ SC_PROBLEM_EX_EASE_OF_REMOVAL This parameter provides ease of removal impact of threat. ■ SC_PROBLEM_EX_STEALTH This parameter provides stealth impact of threat. ■ SC_PROBLEM_EX_SUBCATEGORY_DESCRIPTION ■ SC_PROBLEM_EX_EXTRA_THREAT_INFO The value of this attribute returns true if ICAP response contains extra information about threats (viral, non viral) else false is returned. ■ SC_PROBLEM_EX_INSIGHT_CACHE_DATE This parameter provides the Insight cache revocation date. ■ SC_PROBLEM_EX_INSIGHT_CACHE_REV This parameter provides the Insight cache revocation revision. ■ SC_PROBLEM_EX_INSIGHT_SETTINGS_DATE This parameter provides the Insight settings date. ■ SC_PROBLEM_EX_INSIGHT_SETTINGS_REV This parameter provides the Insight settings revision. ■ SC_PROBLEM_EX_INSIGHT_SYMVT_DATE This parameter provides the Insight SymVT date. ■ SC_PROBLEM_EX_INSIGHT_SYMVT_REV This parameter provides the Insight SymVT revision. ■ SC_PROBLEM_EX_INSIGHT_RESULT This parameter provides the Insight result of the file. ■ SC_PROBLEM_EX_REPUTATION This parameter provides the reputation score of the file. ■ SC_PROBLEM_EX_AGE

Table 2-16 ScanResultGetProblem parameters *(continued)*

Parameter	Description
	This parameter provides the age of the file. ■ SC_PROBLEM_EX_PREVALENCE This parameter provides the prevalence of the file. ■ SC_PROBLEM_EX_FILETYPE This parameter provides the true type of the file. ■ SC_PROBLEM_EX_SHA256 This parameter provides the Insight SHA256 value of the file. ■ SC_PROBLEM_EX_EXTRA_INSIGHT_INFO This parameter provides the Insight information if available. ■ SC_PROBLEM_EX_EXTRA_APK_ RESULT_INFO This value indicates whether APK result code is available in result or not. If the value is true, then APK result code is available otherwise no APK result code is present in result. ■ SC_PROBLEM_EX_APK_RESULT_ID Used to get the APK result code from the result. ■ SC_PROBLEM_EX_INTERNAL_SERVER_ERR_REASON This parameter provides an additional information that is passed with INTERNAL_SERVER_ERROR if available (string). **Note:** SC_PROBLEM_EX_APK_RESULT_ID value will be present in scan result only if one of the below mentioned scenario occurs during APK file scanning. ■ No APK reputation feature license at Symantec Protection Engine. ■ Symantec Protection Engine's APK scanner is overloaded. ■ Some error occurred while APK file scanning.
pszValueOut	Buffer that holds the attribute value being retrieved.
pnValueLengthInOut	Pointer to the variable that holds the size of the pszValueOut buffer. When the function returns, the size of the attribute string is placed in this variable. You can determine if the buffer was large enough to hold the entire string by seeing whether pnValueLengthInOut is smaller after the call than before. (If the value is smaller, the buffer was large enough.)

ScanResultGetProblem return codes

Table 2-17 lists the ScanResultGet Problem return codes (negative values are warnings; positive values are errors).

Table 2-17 ScanResultGetProblem return codes

Value	Return code	Description
5	SC_OUTOFRANGE_PARAMETER	At least one parameter that was passed to the function was out of range.
1	SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
0	SC_OK	Success.

ScanResultGetProblem description

A buffer is supplied to hold the information about the virus, and a pointer is supplied to an integer that holds the size of the buffer. When the function returns, the integer holds the amount of data that was placed in the buffer. If the buffer is not large enough to hold the information, as much information as possible is copied into the buffer. If the value of the integer is the same after the call as it was before the call, the buffer most likely was not large enough to hold the information. See the description of the nAttribute parameter for the type of information that can be retrieved.

SC_DECODE_DISPOSITION

The problem disposition is retrieved with ScanResultGetProblem() in the form of a string. The SC_DECODE_DISPOSITION function is a macro that converts the string to an integer and defines the result codes as integers for easier processing.

```
int SC_DECODE_DISPOSITION( char *pszDisposition )
```

SC_DECODE_DISPOSITION parameters

Table 2-18 lists the parameters that are used.

Table 2-18 SC_DECODE_DISPOSITION parameters

Parameter	Description
pszDisposition	The pszValueOut that is returned from ScanResultGetProblem with nAttribute equal to SC_PROBLEM_DISPOSITION.

SC_DECODE_DISPOSITION return codes

Table 2-19 lists the SC_DECODE_DISPOSITION return codes.

Table 2-19 SC_DECODE_DISPOSITION return codes

Return code	Description
SC_DISP_REPAIRED	The infected file was repaired.
SC_DISP_UNREPAIRED	The infected file was not repaired.
SC_DISP_DELETED	The infected file was deleted.

ScanResultsFree

The ScanResultsFree function is used to free the HSCANRESULT structure. This function must be called when the result structure is no longer needed to free the allocated memory.

```
SC_ERROR ScanResultsFree( HSCANRESULT hScanResult )
```

ScanResultsFree return codes

Table 2-20 lists the ScanResultsFree return codes (negative values are warnings; positive values are errors).

Table 2-20 ScanResultsFree return codes

Value	Return code	Description
0	SC_OK	Success.
-1	SC_NULL_PARAMETER	A parameter that was passed to the function is NULL when it should not be.

ScanClientShutDown

The ScanClientShutDown function is called to clean up after all scanning is complete (for example, before program termination).

```
SC_ERROR ScanClientShutDown( HSCANCLIENT hScanClient )
```

ScanClientShutDown return codes

Table 2-21 lists the ScanClientShutDown return codes (negative values are warnings; positive values are errors).

Table 2-21 ScanClientShutDown return codes

Value	Return code	Description
3	SC_MEMORY_ERROR	A memory allocation error has occurred.
0	SC_OK	Success.

ScanClientSecureShutDown

The ScanClientSecureShutDown function is called to clean up after all scanning is complete (for example, before program termination).

```
SC_ERROR DECLSPEC_SYMCSAPI ScanClientSecureShutDown(HSCANCLIENT hScanClient);
```

Note: ScanClientSecureShutDown API should be called only if ScanClientSecureStartUp was called earlier.

ScanClientSecureShutDown return codes

Table 2-22 lists the return codes (negative values are warnings; positive values are errors).

Table 2-22 ScanClientSecureShutDown return codes

Value	Return code	Description
3	SC_MEMORY_ERROR	A memory allocation error has occurred.
0	SC_OK	Success.

ScanClientStreamStart

The ScanClientStreamStart function is used for scanning streams. It can also be used when you want to accept an input stream for scanning rather than an entire file at one time. For example, if a file is being received through an HTTP stream as a user uploads a file to a Web site, stream scanning can be used.

Note: The OriginalFileName parameter must be Unicode UTF-8 encoded.

```
SC_ERROR DECLSPEC_SYMCSAPI ScanClientStreamStart
(
 HSCANCLIENT hScanClient,
 LPSTR       pszOriginalFileName,
 LPSTR       pszConfigPolicy,
```

```
 HSCANSTREAM *phScanStream
)
```

ScanClientStreamStartEx

If the ScanClientStreamStartEx() function is used (instead of ScanClientStreamStart()),then an extended problem attribute set will be available from Symantec Protection Engine, provided the required parameters are enabled on the server.This extended problem attribute set contains enhanced categorization information about the detected threat.

The ScanClientStreamStart() and ScanClientStreamStartEx() function have same parameters and return codes.

```
SC_ERROR DECLSPEC_SYMCSAPI ScanClientStreamStartEx
(
 HSCANCLIENT hScanClient,
 LPSTR       pszOriginalFileName,
 LPSTR       pszConfigPolicy,
 HSCANSTREAM *phScanStream
);
```

ScanClientStreamStartInsight

If the ScanClientStreamStartInsight() function is used, then Symantec Protection Engine will additionally query Symantec Reputation servers to get the additional reputation result for specified file. The ScanClientStreamStartInsight() function has 2 additional optional parameters compared to ScanClientStreamStart() and ScanClientStreamStartEx() functions.

disableInsightCall specifies whether to enable or disable insight server query. By default disableInsight is 0.

Attributes configured in InsightOptions structure can be used to query the reputation server for file reputation result. Following are the attributes :

- insightAggressionLevel - Insight aggression level used by SPE to convict file using reputation result.
- MD5Hash – MD5 hash of file
- SHA256Hash – SHA256 Hash of file.
- fileSourceIP – Source IP of the file.
- fileSourceURL- Source URL of the file.
- IsDigitallySigned – whether the fie being sent for scanning is digitally signed.

The return codes for ScanClientStreamStartInsight() function are same as ScanClientStreamStart() and ScanClientStreamStartEx() functions.

```
SC_ERROR DECLSPEC_SYMCSAPI ScanClientStreamStartInsight
(
 HSCANCLIENT     hScanClient,
 LPSTR           pszOriginalFileName,
 LPSTR           pszConfigPolicy,
 int             disableInsightCall,
 InsightOptions* insightOptions,
 HSCANSTREAM     *phScanStream
);
```

ScanClientStreamStart parameters

Table 2-23 lists the ScanClientStreamStart parameters that are used.

Table 2-23 ScanClientStreamStart parameters

Parameter	Description
hScanClient	This parameter should be set up using ScanClientStartUp().
pszOriginalFileName	The original name of the file to be scanned as it was named on the user's computer.
pszConfigPolicy	A null-terminated string of the form <option:value>;;;<option:value>... No spaces are allowed. See "ScanClientScanFile" on page 28.
phScanStream	Pointer to an HSCANSTREAM variable.
InsightOptions*	This parameter is used only for ScanClientScanFileInsight.

InsightOptions	Structure of input for insight scanning
	`typedef struct InsightOptions`
	`{`
	`int insightAggressionLevel;`
	`char* MD5Hash;`
	`char* SHA256Hash;`
	`int isDigitallySigned;`
	`char* fileSourceURL;`
	`char* fileSourceIP;`
	`}InsightOptions;`
	This parameter is used only for ScanClientScanFileInsight

ScanClientStreamStart return codes

Table 2-24 lists the ScanClientStreamStart return codes (negative values are warnings; positive values are errors).

Table 2-24 ScanClientStreamStart return codes

Value	Return code	Description
8	SC_CONNPOOLSIZE_EXCEEDED	Maximum persistent connection limit has been reached. No more connections allowed.
7	SC_SERVER_TOO_BUSY	The Symantec Protection Engine server is busy and unable to process the scan request.
6	SC_CONNECT_FAILURE	Attempt to connect to a Symantec Protection Engine failed.
3	SC_MEMORY_ERROR	A memory allocation error has occurred.
2	SC_SOCKET_FAILURE	A socket communication error has occurred.
1	SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
0	SC_OK	Success.
10	SC_SECURE_CONTEXT_INITIALIZATION_FAILURE	Failed to initialize SSL context using provided parameters.
11	SC_SECURE_CONNECT_FAILURE	Failed to securely connect to server.

Setting up stream scanning

The stream scanning feature lets the stream from the user's computer be sent directly to Symantec Protection Engine, rather than first receiving the entire file and then calling ScanClientScanFile().

To set up stream scanning

1. Call ScanClientStreamStart(), ScanClientStreamStartEx(), or ScanClientStreamStartInsight() to initialize the HSCANSTREAM variable.

   ScanClientStreamStart(), ScanClientStreamStartEx(), or ScanClientStreamStartInsight() must be called for each file to be scanned to initialize the HSCANSTREAM variable. HSCANSTREAM variables can be reused only after the stream has been closed with ScanClientStreamFinish() or ScanClientStreamAbort(). The OriginalFileName parameter must be Unicode UTF-8 encoded.

2. Send data to the server in chunks as it is received using ScanClientStreamSendBytes().

3. When all data is sent, call ScanClientStreamFinish().

   To abort the scan between the Start() and Finish() calls, call ScanClientStreamAbort().

ScanClientStreamSendBytes

The ScanClientStreamSendBytes function is used to send chunks of data after HSCANSTREAM has been initialized with ScanClientStreamStart().

See "ScanClientStreamStart" on page 47.

```
SC_ERROR ScanClientStreamSendBytes
(
HSCANSTREAM hStream,
LPBYTE   lpabyData,
DWORD     dwLength
)
```

ScanClientStreamSendBytes parameters

Table 2-25 lists the parameters that are used.

Table 2-25 ScanClientStreamSendBytes parameters

Parameter	Description
hStream	The HSCANSTREAM variable, which must be initialized by a call to ScanClientStreamStart() or ScanClientStreamStartEx().
lpabyData	Pointer to a buffer that contains the next chunk of data to be sent.
dwLength	Size, in bytes, of the next chunk of data to be sent.

ScanClientStreamSendBytes return codes

Table 2-26 lists the ScanClientStreamBytes return codes (negative values are warnings; positive values are errors).

Table 2-26 ScanClientStreamSendBytes return codes

Value	Return code	Description
2	SC_SOCKET_FAILURE	A socket communication error has occurred.
1	SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
0	SC_OK	Success.

ScanClientStreamFinish

The ScanClientStreamFinish function must be called after an entire file has been sent to Symantec Protection Engine to be scanned using ScanClientStreamSendBytes().

See "ScanClientStreamStart" on page 47.

```
SCSCANFILE_RESULT ScanClientStreamFinish
(
HSCANSTREAMhStream,
LPSTRpszOutputFileName,
LPHSCANRESULTSphScanResults
)
```

ScanClientStreamFinish parameters

Table 2-27 lists the parameters that are used.

Table 2-27 ScanClientStreamFinish parameters

Parameter	Description
hStream	The HSCANSTREAM variable, which must be initialized by a call to ScanClientStreamStart(), ScanClientStreamStartEx(), or ScanClientStreamStartInsight().

Table 2-27 ScanClientStreamFinish parameters *(continued)*

Parameter	Description
pszOutputFileName	The storage location for the repaired file. (No output file is created unless the input file is infected and the infection is repairable.) This parameter can be any of the following: ■ A null-terminated string that is a path to where the repaired file is to be stored. ■ A char array at least MAX_STRING long, with the first byte set to '\0'. The API generates a file name for the repair file. When the function returns, pszOutputFileName is set to the name of the repaired file. **Note:** If this parameter is NULL, the API has Symantec Protection Engine scan the file for viruses but not attempt repair. However, this is not the recommended method for forcing Symantec Protection Engine to scan files for viruses but not attempt repair. You should set the pszConfigPolicy to ScanOnly instead.
phScanResults	When the function returns, this handle points at a structure that contains information about any problems (infections) found during the scan. If none were found, this parameter is NULL unless the AlwaysReportDefInfo:1 policy is used in the scan. Information about problems is extracted using the described functions. If this parameter is not NULL, the memory must be released using ScanResultsFree().

ScanClientStreamFinish return codes

Table 2-28 lists the ScanClientStreamFinish return codes (negative values are warnings; positive values are errors).

Table 2-28 ScanClientStreamFinish return codes

Value	Return code	Description
-3	SCSCANFILE_INF_NO_REP	The file was infected with a virus, but no repair was possible.
-1	SCSCANFILE_INF_REPAIRED	The file was infected, and repair was successful.
0	SCSCANFILE_CLEAN	No virus was found.
3	SCSCANFILE_FAIL_ABORTED	The scan was aborted abnormally.
4	SCSCANFILE_INVALID_PARAM	Function was called with an invalid parameter.

Table 2-28 ScanClientStreamFinish return codes *(continued)*

Value	Return code	Description
5	SCSCANFILE_FAIL_RECV_FILE	An error occurred when attempting to receive the repaired file.
6	SCSCANFILE_FAIL_MEMORY	A memory allocation error has occurred.
7	SCSCANFILE_FAIL_FILE_ACCESS	The server could not access the file to be scanned. This error usually occurs for LOCAL scans when the file permissions are wrong or when the file is not in the path that is specified in the LocalFileScanDir parameter on the server. This error can also occur when the API encounters a problem while writing repaired file data that was received from Symantec Protection Engine to the output file.
10	SCSCANFILE_ERROR_SCANNINGFILE	An internal server error occurred while Symantec Protection Engine was attempting to repair the file.
15	SCSCANFILE_ABORT_NO_AV_ SCANNING_LICENSE	No valid license for antivirus scanning functionality is installed.

ScanClientStreamAbort

The ScanClientStreamAbort function is called to abort a scan between calls to ScanClientStreamStart() and ScanClientStreamFinish().

```
SC_ERROR ScanClientStreamAbort
(
HSCANSTREAM hStream
)
```

ScanClientStreamAbort return codes

Table 2-29 lists the return codes (negative values are warnings; positive values are errors).

Table 2-29 ScanClientStreamAbort return codes

Value	Return code	Description
1	SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
0	SC_OK	Success.

Table 2-29 ScanClientStreamAbort return codes *(continued)*

Value	Return code	Description
-1	SC_NULL_PARAMETER	A parameter that was passed to the function is NULL when it should not be.

ScanGetNumConnectErrors

Use this function to determine how Symantec Protection Engine IP addresses were tried when attempting to connect in an HSCANRESULT.

```
SC_ERROR ScanGetNumConnectErrors
(
 HSCANRESULT hScanResult,
 int * nNumIPs
)
```

ScanGetNumConnectErrors parameters

Table 2-30 lists the ScanGetNumConnectErrors parameters that are used.

Table 2-30 ScanGetNumConnectErrors parameters

Parameter	Description
hScanResult	An HSCANRESULT variable returned by ScanClientScanFile() or ScanClientStreamFinish().
nNumIPs	The number of Symantec Protection Engine IP addresses to which the client tried to connect. If the AlwaysReportDefInfo: 1 scan policy option is used, there will always be at least one (blank) incident reported that contains the virus definitions date and version.

ScanGetNumConnectErrors return codes

Table 2-31 lists the ScanGetNumConnectErrors return codes (negative values are warnings; positive values are errors).

Table 2-31 ScanGetNumConnectErrors return codes

Return code	Description
SC_OUTOFRANGE_PARAMETER	At least one parameter passed to the function was out of range.
SC_FILEIO_ERROR	A file I/O error has occurred.

Table 2-31 ScanGetNumConnectErrors return codes *(continued)*

Return code	Description
SC_MEMORY_ERROR	A memory allocation error has occurred.
SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
SC_OK	Success.
SC_NULL_PARAMETER	A parameter that was passed to the function is NULL but should not have been.

ScanGetConnectError

Use this function to find out specific information about a Symantec Protection Engine connection failure or success. When you connect to Symantec Protection Engine, you must supply a buffer to hold the retrieved IP address and a pointer to an integer that holds the port number and error return.

```
SC_ERROR ScanGetConnectError
(
 HSCANRESULT hScanResult,
 int nIPNum,
 LPSTR nIPAddr,
 int *nPort,
 SC_ERROR *nConnectError
)
```

ScanGetConnectError parameters

Table 2-32 lists the ScanGetConnectError parameters that are used.

Table 2-32 ScanGetConnectError parameters

Parameter	Description
hScanResult	An HSCANRESULT variable returned by ScanClientScanFile() or ScanClientStreamFinish().
nProblemNum	Integer specifying which problem entry is being investigated.
nIPAddr	IP address of the Symantec Protection Engine to which the client tried to connect. .
nPort	Port number of the Symantec Protection Engine to which the client tried to connect.

Table 2-32 ScanGetConnectError parameters *(continued)*

Parameter	Description
nConnectError	Integer return value received while connecting to Symantec Protection Engine.

ScanGetConnectError return codes

Table 2-33 lists the ScanGetConnectError return codes (negative values are warnings; positive values are errors).

Table 2-33 ScanGetConnectError return codes

Return code	Description
SC_OUTOFRANGE_PARAMETER	At least one parameter passed to the function was out of range.
SC_FILEIO_ERROR	A file I/O error has occurred.
SC_MEMORY_ERROR	A memory allocation error has occurred.
SC_INVALID_PARAMETER	A parameter that was passed to the function was invalid.
SC_OK	Success.
SC_NULL_PARAMETER	A parameter that was passed to the function is NULL but should not have been.

Appendix A

Using the antivirus API

This appendix includes the following topics:

- About the sample code

About the sample code

The sample codes provided in this chapter of the *Symantec Protection Engine C SDK Guide* is for your convenience. There are two connection formats supported by C API 7.x – Legacy connection format and keep-alive/persistent connection format.

Previously, the C API library created a new connection for every single request/response pair. Now, from version 7.x onwards, the C API library additionally supports ICAP keep-alive/persistent connections, which uses the same TCP connection to send and receive multiple ICAP requests/responses.

A sample program (example.cpp) is included in the Symantec Protection Engine.zip file. This sample program demonstrates how to use the Symantec Protection Engine C API to scan files. This example demonstrates the use of both file-based and stream-based scanning. File-based scanning is enabled by default.

Sample code (using Legacy connection format)

The usage and syntax is as follows:

example <Symantec Protection Engine ip>:<Symantec Protection Engine port> [-api:<0|1|2>] <input file> [<input file>...]

On Unix you should compile the code using code similar to the following:

```
cc example.cpp libsymcsapi.a -lsocket -lnsl -DUNIX
```

On Windows, link to the winsock library. For example:

```
ws2_32.lib
#if defined( WIN32 )
#pragma warning(push,3)
#endif

#include <stdio.h>
#include <string>
#include "symcsapi.h"

#if defined( WIN32 )
#include <windows.h>
#endif

#if defined(UNIX)
#include <string.h>
#endif

// Function Prototypes
void print_prob_info( HSCANRESULTS hResults, int iWhichProblem, int api);
int scanfile( HSCANCLIENT scanclient,
char *orig_name, char *actual_name, int api);

// Usage
void printUsage(char* progName)
{
 printf( "Usage: %s <ipaddress>:<port> [-api:<0|1>]
 <input-file> [<input-file>...]\n", progName );
 printf( "\n-api<0|1> \n");
 printf( "\tUse 1 to scan file with new API's,
provides more information about threat detected.\n");
 printf( "\tUse 0 to scan file with legacy API's.Defaults to 0.\n");
}
int main( int argc, char *argv[])
{
 HSCANCLIENT scanclient=NULL;
 char pszStartUpString[MAX_STRING];
 int i;
 int api = 0;
 int apiSwitchPresent = 0;

#if defined( WIN32 )
 // start up winsock
 WORD wVersionRequested;
```

```
 WSADATA wsaData;
 int err;

 // Load WinSock, request version 1.0.
 wVersionRequested = MAKEWORD(1, 0);
 err = WSAStartup(wVersionRequested, &wsaData);
 if (err != 0)
 {
  return 3;// ERROR
 }

 // Confirm WinSock supports the version we requested.
 if (LOBYTE(wsaData.wVersion) != 1 ||
  HIBYTE(wsaData.wVersion) != 0)
 {
  WSACleanup();
  return 3;// ERROR
 }
#endif // defined( WIN32 )

 if( argc < 3 )
 {
  printUsage(argv[0]);
  return 1;
 }
 if(argc == 3)
 {
  if(strcmp(argv[2],"-api:0") == 0 || strcmp(argv[2],"-api:1") == 0)
  {
   // Optional argument is there but file name is not provided
   printUsage(argv[0]);
   return 1;
  }
  else
  {
   // File name is there but -api switch is not there
 so use legacy API for scanning
   api = 0;
  }
 }
 if(argc > 3)
 {
  if(strcmp(argv[2],"-api:0") == 0)
```

```
 {
  // Use legacy API for scanning
  api = 0;
  apiSwitchPresent = 1;
 }
 else
 if(strcmp(argv[2],"-api:1") == 0 )
 {
  // Use extended API for scanning
  api = 1;
  apiSwitchPresent = 1;
 }
 else
 {
  // Use legacy API.
  api = 0;
 }
}
sprintf( pszStartUpString, "server:%s", argv[1] );

if( ScanClientStartUp( &scanclient, pszStartUpString ) > 0 )
{
 printf( "Error in ScanClientStartUp\n");
 return 1;
}

if(apiSwitchPresent == 1)
 i = 3;
else
 i = 2;
for( ; i<argc; i++ )
{
 printf( "============================\n");
 printf( "Scanning file: %s\n", argv[i] );
 printf( "============================\n");
 scanfile( scanclient, argv[i], argv[i], api );
}

ScanClientShutDown( scanclient );

#if defined( WIN32 )
 if (WSACleanup() == SOCKET_ERROR)
 {
```

```
  // ERROR
 }
#endif

 return 0;
}

/*
** Scans a file
**
** Parameters:
**   scanclient: Symantec Protection Engine client connection
**   orig_name: Original name of the file
**   actual_name: Name of the file on this machine
**
** Returns:
**   1 for success
**   0 for failure
*/
int scanfile( HSCANCLIENT scanclient, char *orig_name,
 char *actual_name, int api)
{
 HSCANRESULTS results=NULL;
 SCSCANFILE_RESULT answer;
 char repair_file[MAX_STRING];
 int numproblems=0;

 // Set up the buffer to accept the repair filename
 repair_file[0] = 0;

#define SCANWHOLEFILE
#if defined(SCANWHOLEFILE)
 // Perform the scan
 if( api == 1)
 {
  // Use extended API for scanning
  answer = ScanClientScanFileEx( scanclient,
   orig_name,
   actual_name,
   repair_file,
   //"Default:1",
   //"ScanOnly:1",
   //"RepairOnly:1",
```

```
    "",
    &results
    );
  }
  else
  {
   // Use legacy API for scanning
   answer = ScanClientScanFile( scanclient,
    orig_name,
    actual_name,
    repair_file,
    //"Default:1",
    //"ScanOnly:1",
    //"RepairOnly:1",
    "",
    &results
    );
  }

#else // SCANWHOLEFILE
  char sendbuff[8 * 1024];

  HSCANSTREAM hScanStream = NULL;
  SC_ERROR   scanClientStreamStartResult  = 0;
  if( api == 1)
         scanClientStreamStartResult =
ScanClientStreamStartEx(scanclient, orig_name, "", &hScanStream);
  else
   scanClientStreamStartResult =
ScanClientStreamStart(scanclient, orig_name, "", &hScanStream);

  if (SC_OK != scanClientStreamStartResult)
  {
    if (scanClientStreamStartResult == SC_SERVER_TOO_BUSY)
     printf( "\nError: Server too busy.\n");

    return 0;
  }

  // Open the file and send it
#if defined (WIN32)
  HANDLE fd = CreateFile( actual_name, GENERIC_READ, 0,
  NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
 if (fd == INVALID_HANDLE_VALUE)
 {
  ScanClientStreamAbort(hScanStream);
  return 0;
 }

 DWORD iChunkSize;
 if (!ReadFile( fd, (void *) sendbuff, sizeof(sendbuff),
&iChunkSize, NULL))
 {
  CloseHandle( fd );
  ScanClientStreamAbort(hScanStream);
  return 0;
 }

 if (iChunkSize < 0)
 {
  CloseHandle( fd );
  ScanClientStreamAbort(hScanStream);
  return 0;
 }

 while ( iChunkSize > 0 )
 {
  if (SC_OK != ScanClientStreamSendBytes(hScanStream,
 (LPBYTE)sendbuff, iChunkSize))
  {
   CloseHandle( fd );
   // Do not call ScanClientStreamAbort here
   return 0;
  }

  // Keep reading the file
  if (!ReadFile( fd, (void *) sendbuff, sizeof(sendbuff),
&iChunkSize, NULL))
  {
   CloseHandle( fd );
   ScanClientStreamAbort(hScanStream);
   return 0;
  }

  if (iChunkSize < 0)
  {
```

```
   CloseHandle( fd );
   ScanClientStreamAbort(hScanStream);
   return 0;
  }
 }

 CloseHandle( fd );
#else // if defined(WIN32)
 int fd = open(actual_name, O_RDONLY);
 if(fd < 0)
 {
  ScanClientStreamAbort(hScanStream);
  return 0;
 }

 int iChunkSize = read( fd,  sendbuff, sizeof( sendbuff));

 if( iChunkSize < 0 )
 {
  close( fd );
  ScanClientStreamAbort(hScanStream);
  return 0;
 }

 while ( iChunkSize > 0 )
 {
  if (SC_OK != ScanClientStreamSendBytes(hScanStream,
 (LPBYTE)sendbuff, iChunkSize))
  {
   close( fd );
   // Do not call ScanClientStreamAbort here
   return 0;
  }

  // Keep reading the file
  iChunkSize = read( fd,  sendbuff, sizeof( sendbuff));
  if( iChunkSize < 0 )
  {
   close( fd );
   ScanClientStreamAbort(hScanStream);
   return 0;
  }
 }
```

```
 close( fd );
#endif // if defined(WIN32)

 SCSCANFILE_RESULT answer = ScanClientStreamFinish(hScanStream,
 repair_file, &results);
#endif // SCANWHOLEFILE

 if( answer > 0 )
 {
  if(answer == SCSCANFILE_FAIL_SERVER_TOO_BUSY)
   printf( "\nError: Server too busy.\n");
  else
   printf( "**** ERROR! Couldn't scan file\n");

  if( results )
  {
   ScanResultsFree( results );
  }
  return 0;
 }

 switch( answer )
 {
 case SCSCANFILE_INF_NO_REP:
  printf( "File is infected and cannot be repaired\n");
  break;
 case SCSCANFILE_INF_REPAIRED:
  printf( "File was infected, and has been repaired\n");
  break;
 case SCSCANFILE_CLEAN:
  printf( "File is clean\n");
  break;
 default:
  printf( "ScanClientScanFile returned an unexpected value\n");
  break;
 }

 // The results structure will be non-null if a virus was detected.
 if( results )
 {
  if( strlen( repair_file ) )
  {
```

```
   printf( "Repaired file saved as: %s\n", repair_file );
  }
  else
  {
   printf( "No repair file generated\n");
  }

  if( ScanResultGetNumProblems( results, &numproblems ) > 0 )
  {
   printf( "Error getting number of problems\n");
   return 2;
  }
  printf( "%s had %d infection(s):\n",actual_name, numproblems );
 }
 else
 {
  numproblems = 0;
 }

 for( int i=0; i<numproblems; i++ )
 {
  print_prob_info( results, i, api);
 }

 // Be sure to free the results when done!
 ScanResultsFree( results );

 return 1;
}

/*
** Prints scan related information
**
** Parameters:
**   hResults: structure containing the scan results
**   iWhichProblem: scan file problem ID
*/
void print_prob_info( HSCANRESULTS hResults, int iWhichProblem, int api )
{
 char attrib[MAX_STRING];
 int attrib_size;
 int iDisposition;
 char* ptrStrTrue = NULL;
```

```
 ptrStrTrue = "true";
 attrib_size = MAX_STRING;
 ScanResultGetProblem( hResults,
  iWhichProblem,
  SC_PROBLEM_FILENAME,
  attrib,
  &attrib_size );
 printf( "File Name: %s\n", attrib );

 attrib_size = MAX_STRING;
 ScanResultGetProblem( hResults,
  iWhichProblem,
  SC_PROBLEM_VIRUSNAME,
  attrib,
  &attrib_size );
 printf( "Virus Name: %s\n", attrib );

 /* In case of legacy API used for scanning,
Threat category for Non Viral threat detection will be sent
  * and SPE sends threat category in ICAP response
 when Category -2 parameter
  * EnableNonViralThreatCategoryResp in configuration.xml is true,
by default it is false.
  *
  * In case of new API's the threat category and
 other extended threat info will be sent when
  * SPE sends the extended threat info in ICAP response only
when Category -2 parameter
  * EnableThreatCategoryInformation in configuration.xml is true.
By defualt EnableThreatCategoryInformation is true
  *
  * Sub category description of a threat will be sent only
when EnableSubCategoryDescriptionResp Category -2 parameter
  * in configuration.xml is set to true and
EnableThreatCategoryInformation is set to true and extended
API's are used for scanning.
  * By default EnableSubCategoryDescriptionResp is false.
  */

 attrib_size = MAX_STRING;
 ScanResultGetProblem( hResults,
  iWhichProblem,
  SC_PROBLEM_NONVIRAL_THREAT_CATEGORY,
```

```
 attrib,
 &attrib_size );
    if( attrib_size > 0)
 {
     printf( "Threat Category: %s\n", attrib);
 }

attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_VIRUSID,
 attrib,
 &attrib_size );
printf( "Virus ID: %s\n", attrib );

attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_FILE_UNSCANABLE,
 attrib,
 &attrib_size );

if(strcmp(attrib,ptrStrTrue) == 0)
 printf("Unscannable: true\n");
else
 printf("Unscannable: false\n");

// Start of extended threat info
//Print Extended threat Info if it is available
if(api ==  1)
{
 attrib_size = MAX_STRING;
 ScanResultGetProblem( hResults,
  iWhichProblem,
  SC_PROBLEM_EX_EXTRA_THREAT_INFO,
  attrib,
  &attrib_size );
 // If SC_PROBLEM_EX_EXTRA_THREAT_INFO is set to true then t
he extra threat info is present
 if(strcmp(attrib,ptrStrTrue) == 0)
 {
  attrib_size = MAX_STRING;
  ScanResultGetProblem( hResults,
```

```
 iWhichProblem,
 SC_PROBLEM_EX_VIRAL_THREAT_CATEGORY,
 attrib,
 &attrib_size );

if( attrib_size > 0)
{
    printf( "Threat Category: %s\n", attrib);
}

// uber categoriess
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_EX_UBERCATEGORIES,
 attrib,
 &attrib_size );

if (attrib_size > 0)
{
    printf( "Uber Categories: %s\n", attrib);
}

// sub cat id
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_EX_SUBCATEGORY_ID,
 attrib,
 &attrib_size );

if (attrib_size > 0)
{
    printf( "Sub Category ID: %s\n", attrib);
}

// cumulative risk rating
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_EX_CUMULATIVE_RISK_RATING,
 attrib,
 &attrib_size );
```

```
if (attrib_size > 0)
{
    printf( "Cumulative Risk Rating: %s\n", attrib);
}

// performance impact
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_EX_PERFORMANCE_IMPACT,
 attrib,
 &attrib_size );

if (attrib_size > 0)
{
    printf( "Performance Impact: %s\n", attrib);
}

// privacy impact
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_EX_PRIVACY_IMPACT,
 attrib,
 &attrib_size );

if (attrib_size > 0)
{
    printf( "Privacy Impact: %s\n", attrib);
}
// ease of removal
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_EX_EASE_OF_REMOVAL,
 attrib,
 &attrib_size );

if (attrib_size > 0)
{
    printf( "Ease of Removal: %s\n", attrib);
}
```

```
  // stealth
  attrib_size = MAX_STRING;
  ScanResultGetProblem( hResults,
   iWhichProblem,
   SC_PROBLEM_EX_STEALTH,
   attrib,
   &attrib_size );

  if (attrib_size > 0)
  {
      printf( "Stealth: %s\n", attrib);
  }
  // subcategory description
  attrib_size = MAX_STRING;
  ScanResultGetProblem( hResults,
   iWhichProblem,
   SC_PROBLEM_EX_SUBCATEGORY_DESCRIPTION,
   attrib,
   &attrib_size );

  if (attrib_size > 0)
  {
      printf( "Sub Category Description: %s\n", attrib);
  }
 }
}
// End of extended threat info
attrib_size = MAX_STRING;
ScanResultGetProblem( hResults,
 iWhichProblem,
 SC_PROBLEM_DISPOSITION,
 attrib,
 &attrib_size );

iDisposition = SC_DECODE_DISPOSITION( attrib );
switch( iDisposition )
{
case SC_DISP_UNREPAIRED:
 printf( "This infection could not be repaired\n");
 break;
case SC_DISP_REPAIRED:
 printf( "This infection was repaired\n");
 break;
```

```
 case SC_DISP_DELETED:
  printf( "The file with this infection should be deleted\n");
  break;
 default:
  printf( "Unknown Disposition\n");
  break;
 }

 attrib_size = MAX_STRING;
 ScanResultGetProblem( hResults,
  iWhichProblem,
  SC_PROBLEM_DEFINITION_DATE,
  attrib,
  &attrib_size );
 printf( "Virus Definitions dated: %s\n", attrib );

 attrib_size = MAX_STRING;
 ScanResultGetProblem( hResults,
  iWhichProblem,
  SC_PROBLEM_DEFINITION_REV,
  attrib,
  &attrib_size );
 printf( "Virus Definitions Revision: %s\n", attrib );
}
```

Index

Symbols

34

A

antivirus scanning
    API libraries 16
    setting policies 29
API functions
    about 20
    sample code 58
    SC_DECODE_DISPOSITION 45
    ScanClientScanFile 28
    ScanClientScanFileEx 28
    ScanClientShutDown 46
    ScanClientStartUp 21
    ScanClientStreamAbort 54
    ScanClientStreamFinish 52
    ScanClientStreamSendBytes 51
    ScanClientStreamStart 47
    ScanClientStreamStartEx 47
    ScanGetConnectError 56
    ScanGetNumConnectErrors 55
    ScanResultGetNumProblems 34
    ScanResultGetProblem 37
    ScanResultsFree 46
API library
    about 16
    compiling 17
    error handling 20
    linking 17

B

backward compatibility 14

C

cache servers 9
client applications
    about 8
    configuring
        with API libraries 16
code samples
    for API library 58
compiling
    API libraries 17
connectors.. *See* client applications

E

error codes.
    API functions 20
error handling 20
exceptions 20

F

functions (API)
    file-based scanning 21
    list of 20
    stream-based scanning 21

I

ICAP
    about 9
    API libraries 16
integration
    custom 8
Internet Content Adaptation Protocol.. *See* ICAP

K

keep-alive connection
    persistent 14

L

linking
    API libraries 17
load balancing
    ScanClientScanFile 28

M

makefile
    Red Hat Linux 19

makefile *(continued)*
Solaris 19

P

parameters.. *See* API functions
Protection Engine
about 7
proxy servers 9

R

return codes.. *See* API functions
risks 10, 12

S

SC_DECODE_DISPOSITION 45
scan policies
setting
API 29
ScanClientScanFile 28, 33
ScanClientScanFileEx 33
ScanClientScanFileInsight 29
ScanClientShutDown 46
ScanClientStartUp 21
ScanClientStreamAbort 54
ScanClientStreamFinish 52
ScanClientStreamSendBytes 51
ScanClientStreamStart 47
ScanClientStreamStartEx 48
ScanClientStreamStartInsight 48
ScanGetConnectError 56
ScanGetNumConnectErrors 55
ScanResultGetNumProblems 34
ScanResultGetProblem 37
ScanResultsFree 46
stream scanning
ScanClientStreamStart() 50
ScanClientStreamStartEx() 50
ScanClientStreamStartInsight() 50

T

threats 10, 12

U

unscannable files 14

W

winsock
initializing 18
shutting down 18, 29